Analog Engineer's Circuit

具有低功耗睡眠模式的智能 DAC LED 偏置电路

Smart DAC *Katlynne Jones*

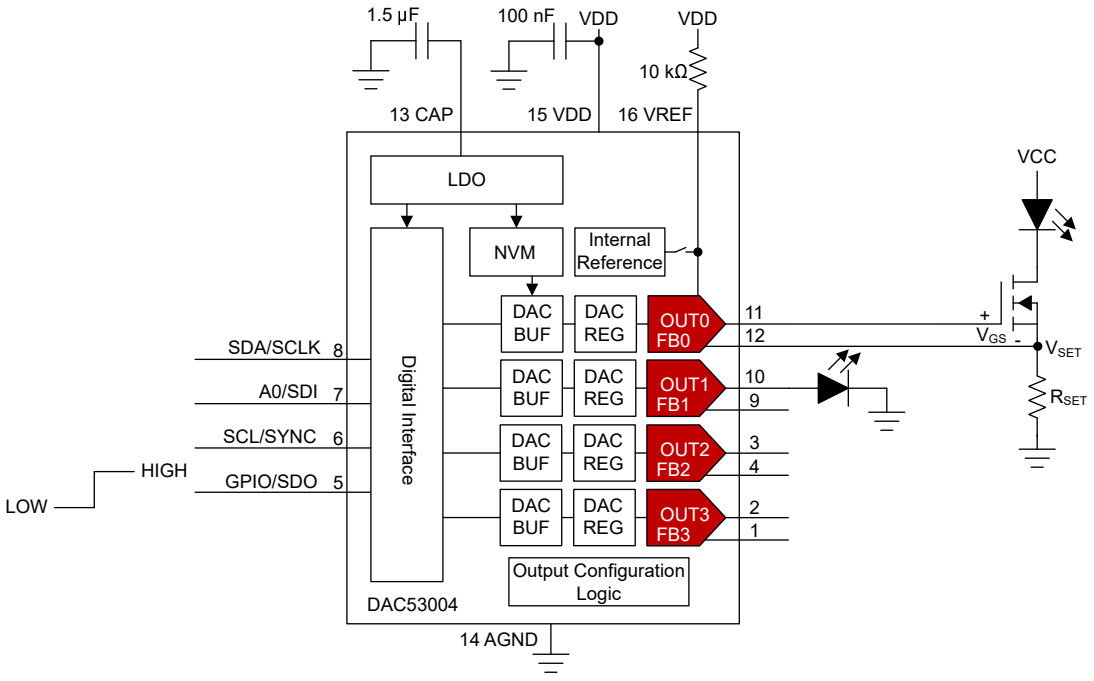
设计目标

主要输入参数	主要输出信号	建议器件
编程 DAC 代码 0x000 至 0xFFF、GPI 触发器的 SPI 或 I^2C 通信	0A 至 250μA 和 0mA 至 20mA LED 电流	DAC53004（10 位）、 DAC63004（12 位）

目标: *使用低功耗智能 DAC 对 LED 进行偏置*

设计说明

此设计使用四通道缓冲电压或电流输出低功耗智能 DAC（如 DAC53004 或 DAC63004（DACx3004））来偏置 LED。在需要几毫安电流的 LED 偏置应用中，智能 DAC 可在强制检测配置下与 MOSFET 相连接。DAC 将设置 MOSFET 的源极电流，并通过改变栅极电压来控制流过 LED（在电源和 MOSFET 漏极之间连接）的电流量。DAC 可以在电流输出模式下使用，以在低电流 LED 偏置应用中使用最高 250μA 电流直接驱动 LED。在使用 MOSFET 配置时，DACx3004 的 V_{FB} 引脚将补偿栅源电压（V_{GS}）压降和 MOSFET 的漂移。DACx3004 具备通用输入输出（GPIO）引脚，支持 DAC 进入和退出深度睡眠模式。可使用智能 DAC 上的非易失性存储器（NVM）保存所有寄存器设置，意味着可在不使用处理器的情况下使用该器件，即使在下电上电后也是如此。此电路可用于条形码扫描仪、条形码读取器、点钞机、POS 打印机、光学模块和电器照明等应用。



设计注意事项

1. *DACx3004 带自动检测型 I2C、PMBus™ 或 SPI 接口的 12 位和 10 位、超低功耗、四通道电压和电流输出智能 DAC* 数据表建议将 100nF 去耦电容器用于 VDD 引脚，将 1.5µF 或更高的旁路电容器用于 CAP 引脚。CAP 引脚连接至内部 LDO。将这些电容器靠近器件引脚放置。
2. 未使用外部基准时，VREF 引脚应通过上拉电阻连接到 VDD。
3. 此示例电路显示了控制 LED 电流的两种方法。可通过 R_{SET} 电阻及使用 DACx3004 输出改变外部 MOSFET 的栅极电压来设置电流，或者可使用 DACx3004 的电流输出模式来设置 LED 电流。
   a. 要通过外部 MOSFET 调整 LED 电流，请选择 R_{SET} 电阻并使用 DAC 输出改变栅极电压。R_{SET} 的计算方法为：

      $$R_{SET} = \frac{V_{SET}}{I_{LED}}$$

      如果选择 0V 至 2.5V 的 DAC 输出电压范围，并且需要的 LED 电流范围为 0mA 到 20mA，则 R_{SET} 计算为：

      $$R_{SET} = \frac{2.5\ V}{20\ mA)} = 125\ \Omega$$

      DAC 代码的计算方法为：

      $$Code = \frac{V_{DAC}}{V_{REF}} \times 1024$$

      在此设计中，内部基准处于断电状态以限制功耗。此配置可补偿因温度、漏极电流和 MOSFET 老化而导致的栅源压降。假定典型栅源电压为 1.2V，电源余量为 200mV，则 DAC 的 VDD 必须至少为 (2.5V + 1.2V + 200mV) = 3.9V。如果使用 5V VDD 作为基准，则 10 位 DAC53004 的高 DAC 值和低 DAC 值变为：

      $$Code = \frac{2.5\ V}{5\ V} \times 1024 = 512\ d$$

      $$Code = \frac{0\ V}{5\ V} \times 1024 = 0\ d$$

      其中
      - d = 十进制数

   b. DAC 可以在电流输出模式下使用，以通过最高 250µA 电流直接驱动 LED。如果选择 ±250µA 范围，则 8 位电流 DAC53004 代码的计算方法为：

      $$Code = \frac{(I_{DAC} - I_{MIN}) \times 256}{I_{MAX} - I_{MIN}}$$

      高 DAC53004 代码和低 DAC53004 代码变为：

      $$Code = \frac{(250\ \mu A + 250\ \mu A) \times 256}{250\ \mu A + 250\ \mu A} = 256\ d$$

      $$Code = \frac{(0\ \mu A + 250\ \mu A) \times 256}{250\ \mu A + 250\ \mu A} = 128\ d$$

      256 十进制数 (256 d) 向下舍入为 255 d，因此高值为 248.04µA。

4. DACx3004 的功耗因使用的配置和断电设置而异。功耗的计算方法为：

$$P = (VDD \times IDD_{SLEEP}) + \sum_{x=0}^{N-1} (VDD \times IDD_X)$$

其中

- IDD_{SLEEP} 是器件在睡眠模式下的静态电流
- N 是通电的通道数
- IDD_X 是每通电通道的静态电流

a. 一个 DAC 通道通过外部 MOSFET 在电压输出配置中的电压输出模式下通电。电压输出模式下的静态电流为每通道 35μA（典型值）。在睡眠模式下，DAC 的静态电流为 21μA（最大值）。VDD 为 5V 时，功耗公式变为：

$$P = (5\ V \times 21\ \mu A) + (5\ V \times 35\ \mu A) = 280\ \mu W$$

此计算不包括从 VCC 流经 R_{SET} 的负载电流。

b. 一个 DAC 通道在电流输出配置中的电流输出模式下通电。当电流输出范围为 0μA 到 250μA 时，静态电流为每通道 18μA（典型值）。在此配置中，还需要加上 DAC 输出通道提供的负载电流。功耗公式变为：

$$P = (5\ V \times 21\ \mu A) + (5\ V \times (18\ \mu A\ +\ 250\ \mu A)) = 1.445\ mW$$

c. 在深度睡眠模式下，所有 DAC 通道处于断电状态，器件静态电流为 3μA（最大值）。功耗公式变为：

$$P = (5\ V \times 3\ \mu A) = 15\ \mu W$$

5. 如果高 DAC 代码值和低 DAC 代码值存储在 MARGIN-HIGH 和 MARGIN-LOW DAC 寄存器中，那么可以对它们之间的压摆率进行编程。压摆时间由 DAC-X-FUNC-CONFIG 寄存器中 SLEW-RATE 和 CODE-STEP 字段的设置确定。压摆时间通过以下方式计算：

$$SleW\ Time = \frac{(MARGIN\_HIGH\_CODE\ -\ MARGIN\_LOW\_CODE + 1)}{CODE\_STEP} \times SLEW\_RATE$$

如果 CODE-STEP 设置为 1 LSB，SLEW-RATE 设置为 4μs/步进，则电压配置的压摆时间为：

$$SleW\ Time = \frac{(512 - 0 + 1)}{1} \times 4\ \mu s = 2.05\ ms$$

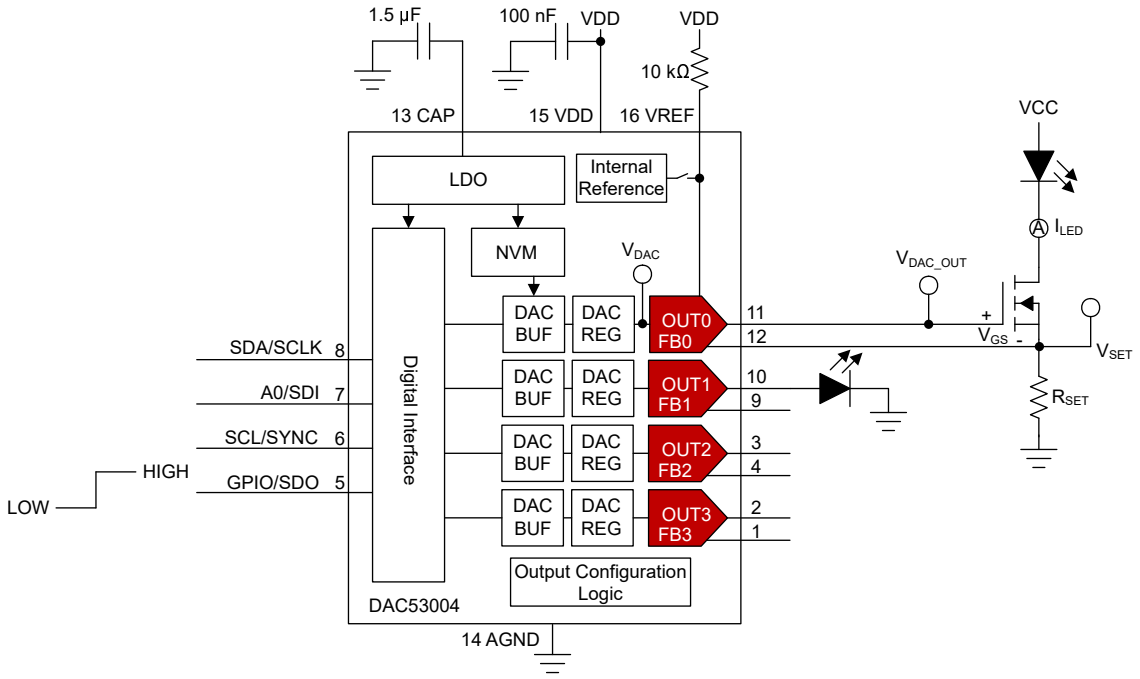
电流输出配置的压摆时间为：

$$SleW\ Time = \frac{(255 - 128 + 1)}{1} \times 4\ \mu s = 512\ \mu s$$

6. GPIO 引脚用作进入和退出深度睡眠模式的输入。GPIO 引脚上的下降沿使器件进入深度睡眠模式。LDO 需要约 550 μ s 时间来关闭，只要 GPIO 输入较低，器件将保持深度睡眠模式。上升沿使器件退出深度睡眠模式。数字电路和 LDO 需要约 550 μ s 时间来打开。寄存器设置部分中介绍了启用 GPIO 以使用此功能的寄存器设置。
7. 可通过寄存器设置部分中说明的初始寄存器设置及使用 I^2C 或 SPI 对 DACx3004 进行编程。可通过将 1 写入到 COMMON-TRIGGER 寄存器的 NVM-PROG 字段来将初始寄存器设置保存到 NVM 中。编程 NVM 后，器件将在重置或下电上电之后，加载具有 NVM 存储的值的所有寄存器。

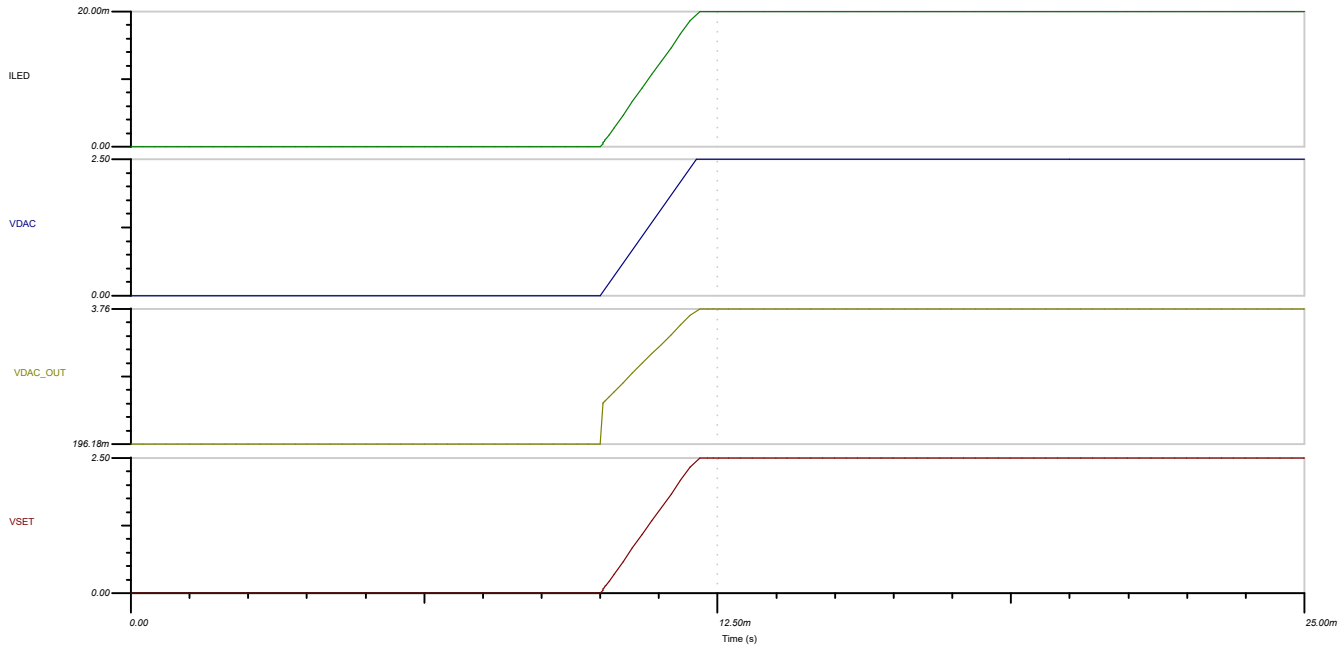
设计仿真

此原理图用于 DAC53004 的以下仿真。



瞬态仿真结果

此仿真显示了当 DAC 从裕度低到裕度高代码转换时的 LED 电流。



English Document: SLAAE28

寄存器设置

电压输出配置的寄存器设置

寄存器地址	寄存器名称	设置	说明
0x01	DAC-0-MARGIN-HIGH	0x8000	[15:4] 0x800：左对齐 10 位数据更新 MARGIN-HIGH 代码
			[3:0] 0x0：不用考虑
0x02	DAC-0-MARGIN-LOW	0x0000	[15:4] 0x000：左对齐 10 位数据更新 MARGIN-LOW 代码
			[3:0] 0x0：不用考虑
0x06	DAC-0-FUNC-CONFIG	0x0001	[15] 0b0：写入 0b1 以将 DAC-0 清除设置设为中标度
			[14] 0b0：写入 0b1 以通过 LDAC 触发器更新 DAC-0
			[13] 0b0：写入 0b1 以使 DAC-0 通过广播命令更新
			[12:11] 0b00：为函数发生器选择相位
			[10:8] 0b000：选择函数发生器生成的波形
			[7] 0b0：写入 0b1 以启用对数转换
			[6:4] 0b000：选择 1 LSB 的代码步进
			[3:0] 0b001：选择 4µs/步进的压摆率
0x1F	COMMON-CONFIG	0x0FF9	[15] 0b0：写入 0b1 以将窗口比较器输出设置为锁存输出
			[14] 0b0：写入 0b1 以锁定器件。将 0b0101 写入 COMMON-TRIGGER 寄存器的 DEV-UNLOCK 字段以进行解锁
			[13] 0b0：写入 0b1 以在地址 0x01 处设置故障转储读取使能
			[12] 0b0：写入 0b1 以启用内部基准
			[11:10] 0b11：将 VOUT3 断电
			[9] 0b1：将 IOUT3 断电
			[8:7] 0b11：将 VOUT2 断电
			[6] 0b1：将 IOUT2 断电
			[5:4] 0b11：将 VOUT1 断电
			[3] 0b1：将 IOUT1 断电
			[2:1] 0b00：将 VOUT0 上电
			[0] 0b1：将 IOUT0 断电
0x20	COMMON-TRIGGER	0x0002	[15:12] 0b0000：写入 0b0101 以解锁器件
			[11:8] 0b0000：写入 0b1010 以触发 POR 复位
			[7] 0b0：写入 0b1 以在 DAC-X-FUNC-CONFIG 寄存器中相应的 SYNC-CONFIG-X 位为 1 时，触发 LDAC 操作
			[6] 0b0：写入 0b1 以基于 DAC-X-FUNC-CONFIG 寄存器中相应的 CLR-SEL-X 位，将 DAC 寄存器和输出设置为零代码或中间代码
			[5] 0b0：不用考虑
			[4] 0b0：写入 0b1 以触发故障转储序列
			[3] 0b0：写入 0b1 以触发 PROTECT 功能
			[2] 0b0：写入 0b1 以读取一行 NVM 进行故障转储
			[1] 0b1：写入 0b1 以将适用的寄存器设置存储到 NVM
			[0] 0b0：写入 0b1 以使用现有 NVM 设置重新加载适用的寄存器

电压输出配置的寄存器设置 (continued)

寄存器地址	寄存器名称	设置	说明
0x24	GPIO-CONFIG	0x4001	[15] 0b0：写入 0b1 以在 GPI 上启用干扰滤波器
			[14] 0b1：启用深度睡眠功能
			[13] 0b0：写入 0b1 以在 GPIO 引脚上启用输出模式
			[12:9] 0b0000：作为输出映射到 GPIO 的 STATUS 功能设置
			[8:5] 0b0000：确定受特定于通道的 GPI 功能影响的通道
			[4:1] 0b0000：选择 GPI 以触发深度睡眠模式
			[0] 0b1：启用 GPIO 引脚的输入模式

电流输出配置的寄存器设置

寄存器地址	寄存器名称	设置	说明
0x01	DAC-0-MARGIN-HIGH	0xFF00	[15:4] 0xFF0：左对齐 8 位数据更新 MARGIN-HIGH 代码
			[3:0] 0x0：不用考虑
0x02	DAC-0-MARGIN-LOW	0x8000	[15:4] 0x800：左对齐 8 位数据更新 MARGIN-LOW 代码
			[3:0] 0x0：不用考虑
0x06	DAC-0-FUNC-CONFIG	0x0001	[15] 0b0：写入 0b1 以将 DAC-0 清除设置设为中标度
			[14] 0b0：写入 0b1 以通过 LDAC 触发器更新 DAC-0
			[13] 0b0：写入 0b1 以使 DAC-0 通过广播命令更新
			[12:11] 0b00：为函数发生器选择相位
			[10:8] 0b000：选择函数发生器生成的波形
			[7] 0b0：写入 0b1 以启用对数转换
			[6:4] 0b000：选择 1 LSB 的 代码步进
			[3:0] 0b001：选择 4µs/步进的压摆率
0x1F	COMMON-CONFIG	0x0FFE	[15] 0b0：写入 0b1 以将窗口比较器输出设置为锁存输出
			[14] 0b0：写入 0b1 以锁定器件。将 0b0101 写入 COMMON-TRIGGER 寄存器的 DEV-UNLOCK 字段以进行解锁
			[13] 0b0：写入 0b1 以在地址 0x01 处设置故障转储读取使能
			[12] 0b0：写入 0b1 以启用内部基准
			[11:10] 0b11：将 VOUT3 断电
			[9] 0b1：将 IOUT3 断电
			[8:7] 0b11：将 VOUT2 断电
			[6] 0b1：将 IOUT2 断电
			[5:4] 0b11：将 VOUT1 断电
			[3] 0b1：将 IOUT1 断电
			[2:1] 0b11：将 VOUT0 断电
			[0] 0b0：将 IOUT0 上电

电流输出配置的寄存器设置 (continued)

寄存器地址	寄存器名称	设置	说明
0x20	COMMON-TRIGGER	0x0002	[15:12] 0b0000：写入 0b0101 以解锁器件
			[11:8] 0b0000：写入 0b1010 以触发 POR 复位
			[7] 0b0：写入 0b1 以在 DAC-X-FUNC-CONFIG 寄存器中相应的 SYNC-CONFIG-X 位为 1 时，触发 LDAC 操作
			[6] 0b0：写入 0b1 以基于 DAC-X-FUNC-CONFIG 寄存器中相应的 CLR-SEL-X 位，将 DAC 寄存器和输出设置为零代码或中间代码
			[5] 0b0：不用考虑
			[4] 0b0：写入 0b1 以触发故障转储序列
			[3] 0b0：写入 0b1 以触发 PROTECT 功能
			[2] 0b0：写入 0b1 以读取一行 NVM 进行故障转储
			[1] 0b1：写入 0b1 以将适用的寄存器设置存储到 NVM
			[0] 0b0：写入 0b1 以使用现有 NVM 设置重新加载适用的寄存器
0x24	GPIO-CONFIG	0x4001	[15] 0b0：写入 0b1 以在 GPI 上启用干扰滤波器
			[14] 0b1：启用深度睡眠功能
			[13] 0b0：写入 0b1 以在 GPIO 引脚上启用输出模式
			[12:9] 0b0000：作为输出映射到 GPIO 的 STATUS 功能设置
			[8:5] 0b0000：确定受特定于通道的 GPI 功能影响的通道
			[4:1] 0b0000：选择 GPI 以触发深度睡眠模式
			[0] 0b1：启用 GPIO 引脚的输入模式

伪代码示例

下面显示了将初始寄存器值编程到 DAC53004 NVM 的伪代码序列。此处给出的值基于在设计注意事项中所做的设计选择。

电压输出配置的伪代码示例

```
1:   //SYNTAX: WRITE <REGISTER NAME (Hex code)>, <MSB DATA>, <LSB DATA>
2:  //Configure GPI for deep-sleep trigger and enable deep-sleep function
3:  WRITE GPIO-CONFIG(0x24), 0x40, 0x01
4:  //Write DAC0 margin high code
5:  //With 16-bit left alignment 0x200 becomes 0x8000
6:  WRITE DAC-0-MARGIN-HIGH(0x01), 0x80, 0x00
7:  //Write DAC0 margin low code
8:  WRITE DAC-0-MARGIN-LOW(0x02), 0x00, 0x00
9:  //Set the CODE-SETP to 1 LSB and SLEW-RATE to 4 µs/step
10: WRITE DAC-0-FUNC-CONFIG(0x06), 0x00, 0x01
11: //Power-up voltage output on channel 0, internal reference disabled
12: WRITE COMMON-CONFIG(0x1F), 0x0F, 0xF9
13: //Enable the GPI, save settings to NVM
14: WRITE COMMON-TRIGGER(0x20), 0x00, 0x02

15: //Trigger for channel 0 margin high
16: WRITE COMMON-DAC-TRIG(0x21), 0x02, 0x00
17: //Trigger for channel 0 margin low
18: WRITE COMMON-DAC-TRIG(0x21), 0x04, 0x00
```

电流输出配置的伪代码示例

```
1:   //SYNTAX: WRITE <REGISTER NAME (Hex code)>, <MSB DATA>, <LSB DATA>
2:  //Configure GPI for deep-sleep trigger and enable deep-sleep function
3:  WRITE GPIO-CONFIG(0x24), 0x40, 0x01
4:  //Write DAC0 margin high code
5:  //With 16-bit left alignment, 0xFF becomes 0xFF00
6:  WRITE DAC-0-MARGIN-HIGH(0x01), 0xFF, 0x00
7:  //Write DAC0 margin low code
8:  //With 16-bit left alignment, 0x80 becomes 0x8000
9:  WRITE DAC-0-MARGIN-LOW(0x02), 0x80, 0x00
10: //Set the CODE-SETP to 1 LSB and SLEW-RATE to 4 µs/step
11: WRITE DAC-0-FUNC-CONFIG(0x06), 0x00, 0x01
12: //Power-up current output on channel 0, internal reference disabled
13: WRITE COMMON-CONFIG(0x1F), 0x0F, 0xFE
14: //Enable the GPI, save settings to NVM
15: WRITE COMMON-TRIGGER(0x20), 0x00, 0x02

16: //Trigger for channel 0 margin high
17: WRITE COMMON-DAC-TRIG(0x21), 0x02, 0x00
18: //Trigger for channel 0 margin low
19: WRITE COMMON-DAC-TRIG(0x21), 0x04, 0x00
```

设计特色器件

器件	关键特性	链接
DAC53004	具有 I2C、SPI 和 PWM 的超低功耗 4 通道 10 位智能 DAC	DAC53004
DAC63004	具有 I2C、SPI、PWM 的超低功耗 4 通道 12 位智能 DAC	DAC63004

使用参数搜索工具查找其他可能器件。

设计参考资料

有关 TI 综合电路库的信息，请参阅*模拟工程师电路手册*。

附加资源

- 德州仪器（TI），DAC63204 评估模块
- 德州仪器（TI），DAC63204 EVM 用户指南
- 德州仪器（TI），高精度实验室 - DAC

要获得 TI 工程师的直接支持，请使用 E2E 社区： e2e.ti.com

重要声明和免责声明

TI“按原样”提供技术和可靠性数据（包括数据表）、设计资源（包括参考设计）、应用或其他设计建议、网络工具、安全信息和其他资源，不保证没有瑕疵且不做出任何明示或暗示的担保，包括但不限于对适销性、某特定用途方面的适用性或不侵犯任何第三方知识产权的暗示担保。

这些资源可供使用 TI 产品进行设计的熟练开发人员使用。您将自行承担以下全部责任：(1) 针对您的应用选择合适的 TI 产品，(2) 设计、验证并测试您的应用，(3) 确保您的应用满足相应标准以及任何其他功能安全、信息安全、监管或其他要求。

这些资源如有变更，恕不另行通知。TI 授权您仅可将这些资源用于研发本资源所述的 TI 产品的应用。严禁对这些资源进行其他复制或展示。您无权使用任何其他 TI 知识产权或任何第三方知识产权。您应全额赔偿因在这些资源的使用中对 TI 及其代表造成的任何索赔、损害、成本、损失和债务，TI 对此概不负责。

TI 提供的产品受 TI 的销售条款或 ti.com 上其他适用条款/TI 产品随附的其他适用条款的约束。TI 提供这些资源并不会扩展或以其他方式更改 TI 针对 TI 产品发布的适用的担保或担保免责声明。

TI 反对并拒绝您可能提出的任何其他或不同的条款。

邮寄地址：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265